ASpect: A New Spectrum and Line Analysis Package for IRAF

Project Status: August 1994

Stephen J. Hulbert, STScI

1.0 Introduction

We are developing general spectral analysis software that will run as an external package under the IRAF environment. This package provides tools to manipulate, display, combine, and model spectral data obtained from a variety of ground- and space-based telescopes and spectrographs. The package will provide a graphical user interface (GUI), using the IRAF Object Manager, to support spectral analysis and to facilitate the efficient management of large and varied spectral data sets and the processing of large quantities of multispectral data.

2.0 Project Specifications

ASpect has several functional requirements; it will be able to:

- Operate on spectra from a wide variety of ground-based and space-based instruments, spanning wavelengths from radio to gamma rays
- Accommodate non-linear dispersion relations
- Provide a variety of functions, individually or in combination, with which to fit spectral features and the continuum
- Perform a single fit to features that are common to more than one input spectrum
- Support input error vectors, including masks for known bad data, but should be able to derive some estimate of the fitting error, even if an input error vector is not available
- Propagate uncertainties throughout the calculations
- Provide a powerful, intuitive user interface to handle the burden of data input/output (I/O), on-line "help", selection of relevant features for analysis, plotting and graphical interaction, and data base management, all in a easy and comprehensible environment
- Run as an external IRAF package

3.0 Project Status

The specifications listed in the previous section have been group decided into four independent areas of development: GUI, internal data structures, analysis algorithms, and documentation. Specific activities over the past year are described in the following sections.

3.1 GUI

To date, a significant part of our effort has been in the area of GUI development. Our initial plan was to develop the interface in the existing (at the time) terminal based graphics facility of IRAF. Once work began, it became evident that the full scope of our analysis needs could not be met using the existing interface. Consequently, we began searching for a suitable graphical interface environment that would still be able to "run" out IRAF tasks. We were able to present our plan at the 1992 Astronomical Data Analysis Software & Systems (ADASS) meeting and discuss with Doug Tody the lack of a good GUI for IRAF. During the subsequent year we selected the X windows, public-domain software, tcl, and its associated widget set, tk, in which to develop our GUI. We were able to demonstrate our working prototype GUI at the 1993 ADASS meeting. Surprisingly, the IRAF group demonstrated a new X windows-based Object Manager that permits IRAF developers to create tasks that used windows! So, in March 1994 we invited Doug Tody to Baltimore to discuss in detail this new Object Manager capability in IRAF. Eventually, we decided to cease our independent efforts in GUI development and switch over to the Object Manager. Since then, we have "started over" in creating our GUI and associated tasks solely under the IRAF system. This decision allows us to meet one of our original design goals, namely that *ASpect* will exist as an add-on package to IRAF. We will demonstrate the latest version of *ASpect* at the upcoming (late September) 1994 ADASS meeting. Figure 1 shows a view of the main window of the current version of *ASpect* and Figure 2 shows a file requester window.

Additional details on the GUI interface to the individual *ASpect* tasks are contained in the attached document, *ASpect* Software Design.

3.2 Internal Data Structures

The basic structure for keeping track of data with *ASpect* is a STSDAS binary table. This concept was part of our original design and continues to meet our needs. The current version of *ASpect* can read and write data to and from the *ASpect* format tables. Additional work is needed to enable *ASpect* to be able to read "foreign" data structure as was originally planned. We have determined that the problem of reading generic X-ray data is a difficult one as it may involve more data manipulation than we had anticipated.We have assigned this a lower priority than other "foreign" data structures and will complete this work if time permits.

Details of the internal data I/O are described in the attached document, "*Aspect* Software Design: Database I/O".

3.3 Analysis Algorithms

We have begun work on the various functional tasks of *ASpect* in the past few months. The development of analyses algorithm has progressed to the point where *ASpect* can:

- fit more than one feature in more than one spectrum
- support a variety of fit functions (e.g. linear continuum, power law continuum, broken powerlaw continuum, absorption edge, gaussian absorption line, gaussian emission line, lorentian line profile, and optically thin recombination continuum)
- provides chi-square minimization with a choice of fit algorithms: Marquadt or simplex
- permits constrained fitting of features
- provides estimation of uncertainties for fitted parameters using chi-square as a goodness-of-fit measure

The work in this area has really just started—some areas have yet to be begun. Functional capabilities yet to be added include: spectrum arithmetic and error propagation.

3.4 Documentation

We have developed a prototype html version of the help that runs under Mosiac and lynx. See the attached document, "*ASpect* Software Design: On-Line Documentation" for a description. There is much work to be done on providing good documentation for *ASpect* users.

4.0 Project Schedule

Table 1 shows the original schedule we established to meet the goals to develope the *ASpect* package. As indicated in the tables we have completed about 60% of the project timeline. But as planned we have much work to do in the area of GUI, analysis algorithms, and documentation.

Project Component	Quarter							
	1st	2nd	3rd	4th	5th	6th	7th	8th
Graphical User Interface	○	○	●	●	○	○	●	●
Internal Data Structure	○	○	●	●				
Analysis Algorithms			○	○	●	●	●	●
Documentation			●	●	●	●	●	●
Testing			●	●	●	●	●	●

TABLE 1. Software Development and Implementation Schedule. Symbols in the table have the following meanings: ○ = Develop; ● = Implement; Gray = Completed; White = To Be Completed.

Two years have elapsed since the *ASpect* project began and during this time we have completed about 60% of the project, i.e. we have slipped at least 9 months behind schedule. We attribute this slip several things, the most significant of which is increased functional work in support of the HST mission, in general, and the First Servicing Mission, in particular. Also, since we proposed this project, two of us (Steve Hulbert and Zolt Levay) have taken on new functional assignments which have added to the time pressures. Additionally, the decision to adopt the IRAF Object Manager has cost us much time in the development of our GUI. We believe that this project can be completed in one additional year; we have requested a one-year no-cost extension via the appropriate channels.

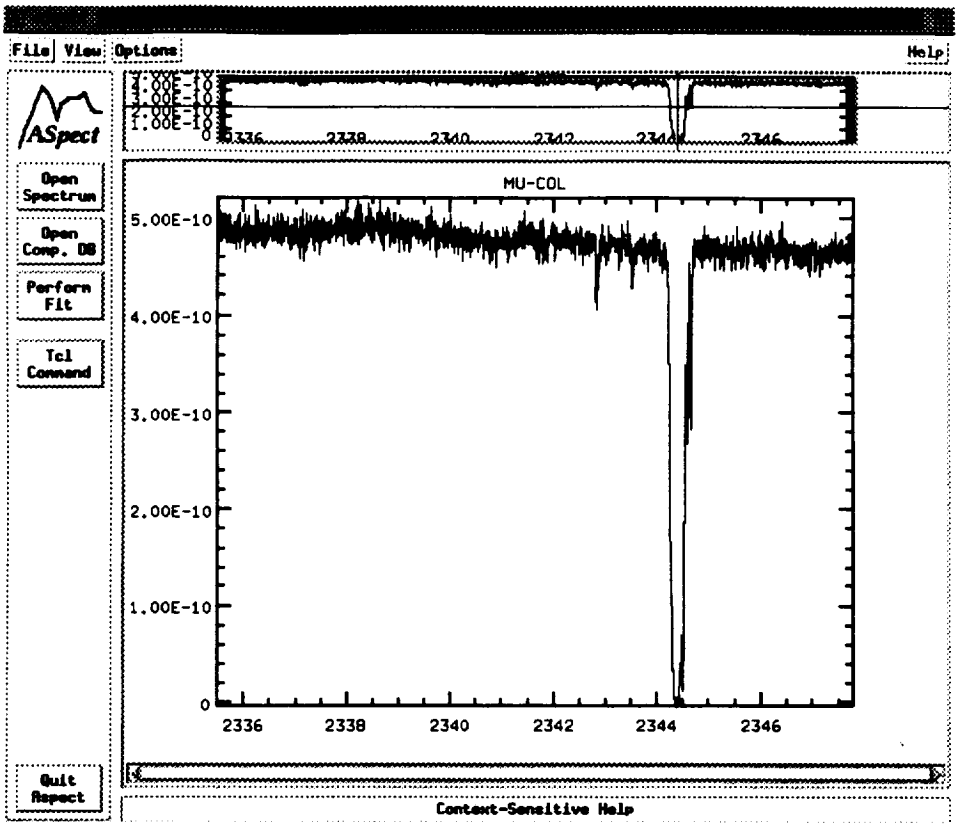


FIGURE 1. Main window for *ASpect* task

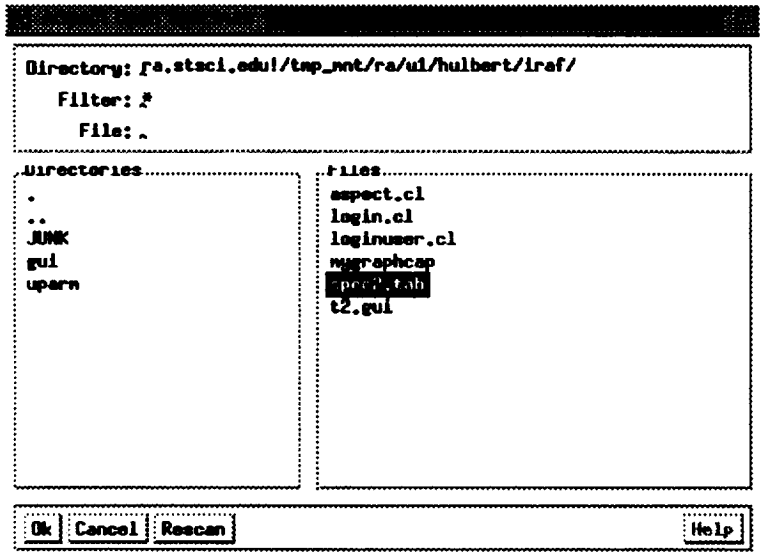


FIGURE 2. File requester window for the *ASpect* task.

ASpect Software Design: Database I/O

Dick Shaw, STSci

1.0 Introduction

The disk data structure used for the ASpect DataBase is STSDAS binary tables. This data structure is accessed through the AS_dbio interface, in the routines found in "dbio.x", "as_db.x", and "as_dbh.x". The routine names are intended to give some idea of their functionality: generally they are prefixed with "db_", although operations on a specific component would be prefixed with "dbc_", those on header parameters with "dbh_", those on a particular parameter with "dbp_", and so on. See the "as_db.h" include file for table header and column definitions.

2.0 DataBase Access

Below are the descriptions and calling sequences for the ASpect DataBase interface. The routines that are callable from the event loop are:

```
db_read (tp, cp, ic, ip)

db_write (tp, cp, ic, ip, ncp, new)

ncomp = db_index (tp, colptr, ic, ip, nrow)
```

The routines that are callable from the above DBIO routines are:

```
value = dbh_get[i] (tp, key)

dbh_gett (tp, key, index, val, maxchar)

dbh_set[i] (tp, key, val)

dbh_sett (tp, key, index, val)

dbc_add (tp, colptr, cmp, ic, ip)

dbc_get (tp, colptr, ic, npar, c_name, p_name, val, hi,
low, step, ftol, sig, cnstr)

dbc_delete (tp, colptr, ic, ip)

db_init (tp, colptr)

db_col_def (tp, colptr, colname, colunit, colfmt,
cdtype, lendata)

dbc_update (tp, colptr, cmp, ic, ip)
```

3.0 Component Description

See "component.h" for a list of the valid component types, and the names of their associated parameters. A routine is provided to ensure that the component parameters specified in a database table are valid and complete:

```
status = cp_validate (npar, indx, val, hi, lo, stp,
          tol, sig, cnstr, c_name, p_name)
```

The character string "p_name" contains the names of the parameters, and is dimensioned [SZ_PNAME,ARB] in the routine. The integer index corresponding to the component is "cindx". The returned status is "0" if all parameters are present and valid. A value of "-1" indicates that not all required parameters are present, and a positive value gives the index of the first unrecognized parameter.

4.0 DataBase Descriptors

Various meta-data that describe the attributes of the entries in the DataBase table are stored in the table header as FITS-style keywords. They are given below, with example values:

```
N_COMPON=          8                  /Number of components in table
SW_VERSN= 'ASpect Vx.x: 24-FEB-94'    /ASpect software version
ASRANGnn= '4600.0-4900.0'             /Abscissa range for sol'n 'nn'
FLUXUNIT= 'ergs/cm^2/s'               /Flux units
DSPRUNIT= 'Angstroms'                 /Dispersion units
SPECFILn= 'G191B2B.tab'               /Filename for spectrum 'n'
```

ASpect Software Design: On-Line Documentation

Steve Hulbert, STScI

1.0 Introduction

We have implemented a prototype on-line help interface using html that will run under Mosaic and Lynx. This help system will be operate in conjunction with the on-line help available under IRAF as well as the hardcopy versions of the help files available with the *ASpect* software.

2.0 Hardcopy Help

A hardcopy version of all help files and user guides will be available with the distributed software and are required deliverables.

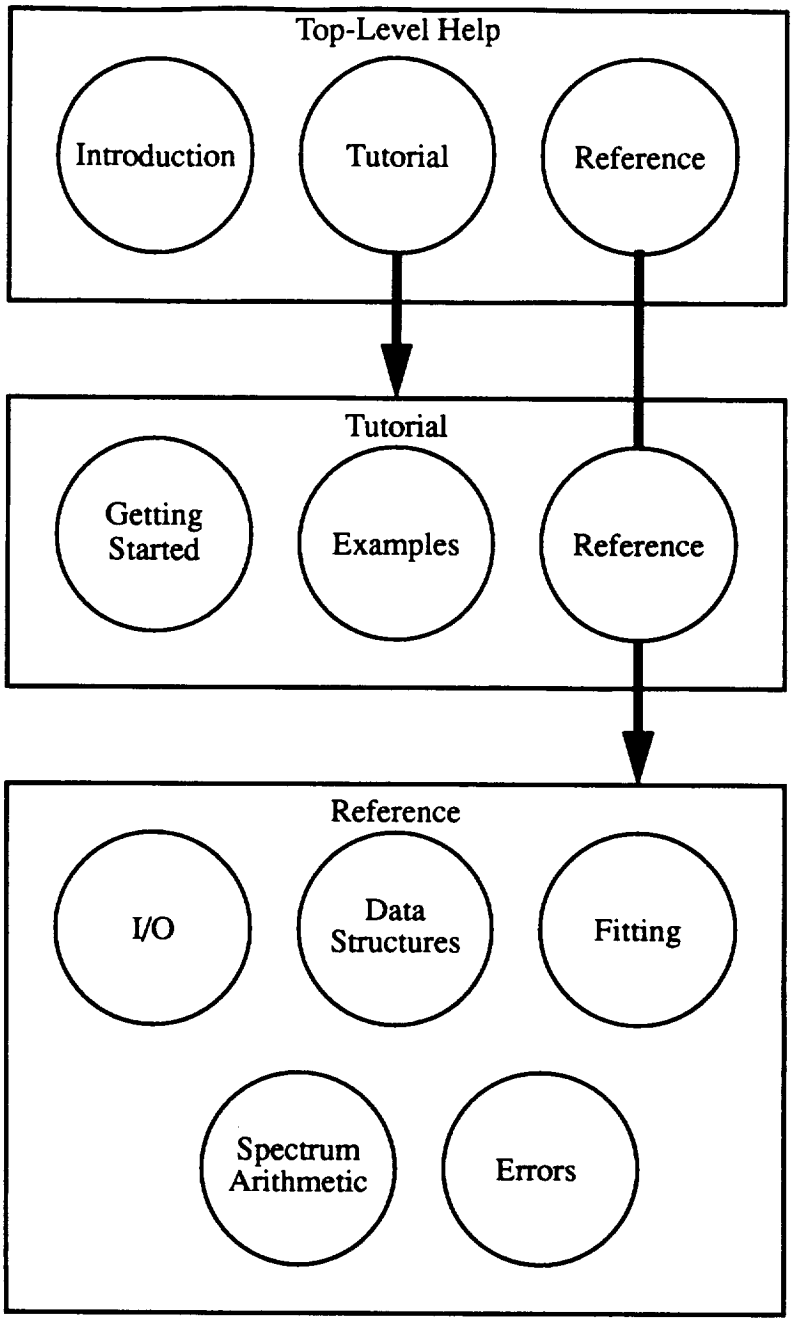
3.0 IRAF Help

On-line help will be available under IRAF using the IRAF help facility. With the current implementation of window-based tasks, IRAF help will not be accessible while the task is running. The IRAF help should be most useful prior to starting up the *ASpect* task and/or while using the noninteractive versions of the *ASpect* tasks. This help is available from the command line (cl).

4.0 GUI Help

The on-line help available while running *ASpect* will be reached by pressing a "Help" button in the GUI. This action will bring up a Mosaic window with the *ASpect* help displayed. The help will be an html version of the IRAF and hardcopy help files. See Figure 1 for the organization of major topics and links. A working model of this hypertext-based help has been created.

FIGURE 1. Major Links in Hypetext Version of On-Line Help



ASpect Software Design

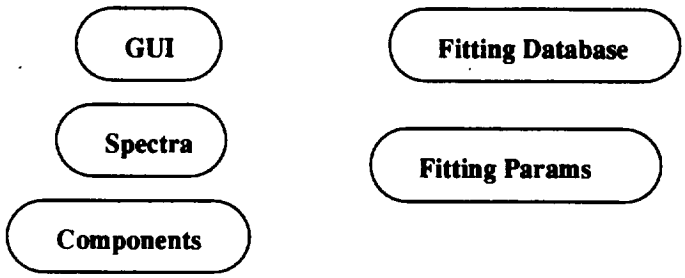
1.0 General Overview

Since we have changed to trying to use the IRAF Object Manager (OBM), the need to create a more formal definition of the software is great. Since the task is required to be monolithic, the internal data structures become very important. Also, interaction with the GUI has "left our hands". This is an attempt at analyzing what needs to be written and the general aspects of the problem. Note that what is described is implemented in task t2.

1.1 The Objects

The objects involved are as follows:

FIGURE 1. *ASpect* Objects



Currently, Dick has developed a prototype Fitting Database interface. He has also done some spectra object definitions, but we haven't seen these yet. Zolt is in charge of developing the GUI itself and he has asked me to develop the GUI Object interface for the task.

1.2 GUI Object

The GUI object needs to handle information passed between the GUI and the task. For the OBM, there are two mechanisms defined for GUI interaction:

1. To the GUI, the task has the `gmsg` and UIParameter interface.
2. From the GUI, the task has the `clgcur` call.

The GUI object primarily consists of the *Event Loop*, which handles events from the GUI. The other part is primarily sending `gmsg` calls to set UIParameter objects. For both directions, an interface, or set of acceptable commands, must be developed. This interface will handle a set of classes of commands. For example, setting components of a fit is one class, file manipulation (reading/writing spectral files) is another class, and performing/viewing fits is a third class. All classes require a "round-trip", i.e. the GUI tells the task that something needs to be done, the task does it which then reports what has occurred back to the GUI.

The classes of actions first need to be defined before the interface can be developed.

1.2.1 GUI Action Classes

The possible actions from/to the GUI are as follows:

1. Select files (spectra, fitting database, etc.)
2. Selecting fitting parameters.
3. Performing fits.
4. Viewing spectra/fits

Unfortunately, all of these have to be developed in unison. The initial goal is to read in a spectrum, view it, choose an object to fit, and display the results. Go for it.

1.2.2 UI Parameters

Initial explorations have indicated that you pretty much want a separate parameter for each piece of information to be kept track of by the GUI. The UI parameter concept can be extended to be a bit more general, i.e. through the call back of the UI parameter, the task can do most anything. However, in discussion with Zolt, it is clear that using UI's for more than single parameter status values can be dangerous.

The implication is that there can be no general UI parameter "object" from the task's viewpoint. However, general libraries can be setup that are "ui", "science", and "utility" related. The code layout is discussed in Section 2.0, "Code Construction," on page 3.

Now that the negative comments are out of the way, the UI parameters relating to fitting components can be generalized. Specifically, the "value" sent to the UI would take the form:

type parameter value

where:

type = The type of feature, i.e. gaussian

parameter = The parameter of the feature

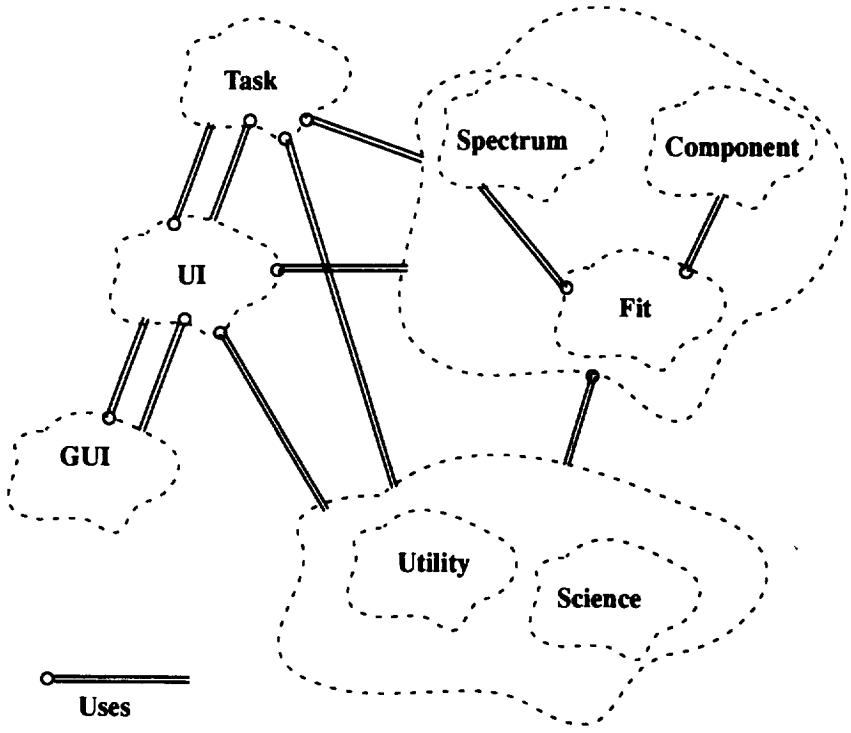
value = Value of the parameter.

Similarly, if the GUI is changing components/features, the messages sent to the task would take a similar form, though a message may contain more than one such change. The subroutines intercepting these message (by both the GUI and task) would parse out the value and take appropriate action, such as the GUI updating a label for graphic indication of the fit.

2.0 Code Overview

The code libraries will take the form matching closely the defined objects and the actions performed on those objects. Code interaction between the GUI and the rest of the task will take the following form:

FIGURE 2. *ASpect* Object Relationships



An explanation of the objects and associated code libraries are described below.

2.1 GUI Object

Currently, this is the IRAF Object Manager from which the visible interface is built.

2.2 The UI Object

The UI object is the set of procedures that interface between the GUI and the rest of the task. These procedures will be referred to as `ui` routines and will have names that begin with `ui_`. To be specific, any procedure which uses the `clgcur`, `gmsg`, or any `GIO` routine will be a `ui` routine. No other procedures should do any user-level I/O except for `ui` routines. In general, only task-level routines (see below) will call the `ui` routines. In this way, the lower level routines and objects will remain independent of the user interface in use.

2.3 The Task Object

The Task object are those routines that implement the task functionality. In particular, the main procedure and all global variables are maintained here. Though the specific interactive task for *ASpect* is discussed in this paper, any task written for the *ASpect* project can be used. The naming convention for these procedures depend on what the task name is. For example, the main interactive task for *ASpect* is called `aspect`, thus the `task` routines will be begin with `as_`.

The `task` routines will be called by the `ui` routines and will also call `ui` routines. Also, the `task` routines will have access to all other objects and libraries necessary to carry out the designated functionality.

2.4 The Spectrum Object

The spectrum object simply defines what a spectrum is.

2.5 The Component Object

A fit is made of *components*. A component is a feature of a spectrum to be fit. A component consists of a type and various parameters.

2.6 The Fit Object

The Fit object are the routines that perform the fitting.

2.7 The Science Library

This is the set of basic science routines, including fit functions, etc.

2.8 The Utilities Library

The set of generic routines that don't fall into any of the above categories.

3.0 Interface Specification: GUI Object

This chapter is a detailed description of the interface between the GUI Object and the client. As specified by the OBM, the interface between the OBM and the client are messages sent between the two. There are two types of messages sent from the OBM to the client: single cursor keystrokes called *gkeys*, and *colon commands*. The client uses the `clgcur` routine to read either of these messages. What the client sees is as follows:

1. Cursor Location: The X, Y coordinates of the mouse when the event was sent.
2. The GIO WCS that is pertinent to the X,Y coordinates sent.

3. The key which initiated the event.

4. If a colon command, the string sent with the colon key event.

In general, gkey events are used to perform an action based on the cursor position and colon commands initiate actions that may or may not need the cursor position.

The client communicates with the GUI by sending messages to *UIParameters*. UIParameters are GUI objects whose specific purpose is to maintain the state of the client in the GUI. The messages are strings of arbitrary contents. The actions a UIParameter can take upon receiving a message are also arbitrary.

The rest of this chapter defines the messages that can be passed between the GUI and *ASpect*.

3.1 GKey Events

The following gkey events are defined. The ASCII key of each event is listed with an explanation of the client's behavior upon receipt of the particular event.

- **q**

  The quit event. The client task will make sure that all work has been saved and then terminate. If there are items that the user may want to save, the client will inform the GUI. If not, the client will tell the GUI that all is done so that the GUI itself can terminate.

3.2 Colon Commands

The follow colon commands are defined. In general, the first word of the string is the *command*, followed by a list of parameters the client will need to execute the command.

- **chdir *directory***

  Set the default working directory to *directory*. The client subsequently will send messages to the `uiCurDir` and `uiSubDirs` UIParameters containing information about the new directory.

- **db_open *file***

  Open *file* as a fit database and read in the fit components.

- **db_save *file***

  Save the currently defined fit components as a fit database in the specified *file*.

- **fit_execute**

  Perform a fit using the current parameters.

- **fit_niter_set *max***

Set the maximum number of iterations the fit can perform to *max*.

- **fit_range_add *gterm x1 x2***

  Add the range *x1*, *x2* to the list of ranges to be fit. *x1*, *x2* are specified in NDC coordinates relative to the specified *gterm*.

- **fit_range_del *gterm x***

  Delete the range which contains *x* from the list of fit ranges. *x* is specified in NDC coordinates relative to *gterm*. If *x* lies within multiple ranges, the smallest range will be deleted.

- **fit_type_set *type***

  Set the fitting algorithm to *type*. The currently available types are `simplex` and `marquadt`.

- **fit_tol_set *tolerance***

  Set the fitting tolerance to *tolerance*.

- **gt_change *gterm***

  Set the active gterm to *gterm*. **Not implemented yet.**

- **gt_work_redraw *left right bottom top gterm***

  Redraw the plot in the work gterm, using the limits specified in *left*, *right*, *bottom*, and *top*. The limits are specified in NDC coordinates relative to the indicated *gterm*.

- **ls *filter***

  Get a directory listing using the specified file *filter*. The client will send a message to the `uiFiles` UIParameter containing the list of files in the current directory.

- **sp_open *file***

  Open the specified *file* as a spectrum.

- **sp_save *file***

  Save the current spectrum in the specified *file*. **This is not implemented yet.**

3.3 UIParameters

The following UIParameters have been defined. The UIParameter is listed along with the message that will become the UIParameter's value.

- **uiCurDir *directory***

  The current working directory is *directory*.

- **uiError *message***

An error has occurred in the client, with an explanation in the *message*.

- **uiFiles** ***files***

  files is a space-separated listing of the files in the current directory.

- **uiOpenStat** ***status***

  If a file is being opened, the client informs the GUI about the state of the operation through status. The value of *status* can be as follows:

  - **isdir** ***directory*** - The file to open was actually a directory. The name of the *directory* is specified.
  - **done** - The file was successfully opened.

- **uiSetDir** ***directory***

  The client wants the GUI to set the current directory to *directory*.

- **uiSetHScroll** ***x1 x2***

  Set the horizontal scroll bar such that the thumb extends from *x1* to *x2*, given in NDC coordinates.

- **uiSetWorkViewMarker** ***left right bottom top***

  Set the work view marker in the fullPlot Gterm window to have the specified corner locations, specified in NDC coordinates.

- **uiSubDirs** ***files***

  The current directory contains space-separated list of subdirectories specified in *files*.

- **uiSwitchGterm** ***gterm***

  Set the active Gterm to *gterm*.